Detecting contextual hate speech code words within social media

Jherez Taylor, Prof. Yi-Shin Chen
Co Advisor: Prof. Chaur-Chin Chen
Institute of Information Systems and Applications
National Tsing Hua University, Hsinchu, Taiwan
{jherez.taylor}@gmail.com

Abstract
While relatively rare in face–to–face interactions, social media platforms have recently seen an increase in the occurrence of hate speech discourse. Most methods rely on word blacklists and other text level features such as *n*-grams. While this approach is effective for flagging hate speech content, the discourse is not limited to a specific vocabulary as users are constantly adopting new terms.

In this work we develop a graph based approach that incorporates conventional word window contexts along with syntactic dependency contexts in order to learn the hidden meaning of hate speech code words that have relatively unknown associations to hate speech.

Our proposal utilizes the different types of contexts in which words are utilized with the goal being to identify new code words, thus expanding the hate speech lexicon and improving the accuracy of future classification systems.

Keywords: ***Hate speech, NLP, Twitter, PageRank***

I. Introduction

Merriam-Webster defines a *Code Word* as *a word or phrase that has a secret meaning or that is used instead of another word or phrase to avoid speaking directly*. Hate Speech [HS] can be difficult to define as there are some who argue that restrictions on what constitutes HS are in fact violations of the right to free speech; for this reason it is important to adhere to a rigid definition of HS.

For this work we rely on the definition from the *International Covenant on Civil and Political Rights, Article 20 (2)* which defines Hate Speech as *any advocacy of national, racial or religious hatred that constitutes incitement to discrimination, hostility or violence* [1]. Instances where a given document is objective or makes reference to terms typically associated with HS cannot be considered HS.

The wide scale conditioning of groups of society to hate each other can have disastrous effects as the world observed in the 1930s and 40s. While social networks, particularly Twitter, seek to promote free speech, they also work towards policing content that might be considered HS. The spread of HS can have damaging effects on both the victims and the social network itself. Twitter has reportedly lost business partially as a result of potential buyers raising concerns about the reputation that the social network has for bullying and uncivil communication. [2]. Additionally, governments are seeking to impose hefty fines on social media networks that fail to remove flagged hate speech content within a specified time frame [3] and even holding personal staff accountable for the inaction of these companies. However, the systems in place are not perfect as individuals and groups are constantly applying new linguistic means of bypassing these systems. These means include intentional misspellings and creating code words by adapting common words to have alternative meanings.

In spite of the social and financial damage caused by the proliferation of hate speech across the Internet, research on the topic has been limited due to the subjective nature of identifying hate speech. A few reasons for this are that hate speech discourse is not limited to a specific vocabulary, the lack of standardized evaluation dataset, and the continuous adaptation of hate speech code words from standard words.

This study aims to build a hate speech code word identification method that uses word dependencies in order to detect the contexts in which words are utilized so as to identify new hate speech *code words* that might not exist in the known hate speech lexicon. Specifically, this paper has the following contributions:

- As a means of dealing with the changing nature of language and the constant introduction of new hate speech *code words* we propose the use of a *dependency2vec* neural embedding model that learns functionally similar words.
- We make use of PageRank as a bootstrapping method to identify alternate words that share some relation to other known hate speech words.
- Finally, we develop a contextual word enrichment model to learn out-of-dictionary hate speech code words, thus expanding the lexicon.

Our results show the benefit of considering syntactic dependency-based word embeddings for finding words that function similar to known hate speech words (code words). We present our work as an online system that continuously learns these dependency embeddings, thus expanding the hate speech lexicon and allowing for the retrieval of more tweets where these code words appear.

Reader advisory: We present several examples that feature hate speech and explicit content. We want to warn the reader that these examples are lifted from our data set and are featured here for illustrative purposes only.

II. Related Work

The last several years has seen an increase in research related to identifying HS within online platforms. Warner and Hirschberg [4] produced the cornerstone work in the domain. The authors proposed a supervised approach that labels HS by identifying stereotypes used in text. They hypothesized that HS consists of well-known stereotypes that can then be used to distinguish one form of HS from another. They concluded that creating a language model for each stereotype is a necessary prerequisite for building a model for all HS. To this end, they created a language model based on annotated Antisemitic content which was then used for correlating the presence of HS in anti-Muslim and anti-African HS categories. Their data consisted of text crawled from websites flagged as being producers of HS along with comments from Yahoo! that were flagged by users as being offensive. These comments lacked topical context and were short, typically around 30 words. The authors also reasoned that text which featured evasive behaviour such as intentional misspellings are positive indicators of offensive speech. Language models are effective but they can be costly to build and maintain; as such, we do not utilize this approach in our work. The role of subjectivity and emotion was noted in the conclusion and future work sections of several of the works mentioned thus far. Newer works began to incorporate sentiment analysis and subjectivity detection including Gitari et al. [5], one of the first works with these considerations. The authors developed a classifier that utilizes sentiment analysis and subjectivity detection to determine whether a given sentence is subjective along with its polarity. They differentiated their model by first building a HS lexicon from subjective sentences within the corpus, avoiding the creation of categories on the basis of annotation; resulting in one of the only fully unsupervised approaches within the literature. We initially explored the idea of incorporating sentiment level features into our work but the experiments did not yield significant improvements. As a result, we ultimately decided to adopt the idea of lexicon building instead.

Central to the problem that we attempt to solve is the idea of supplementing the traditional bag of words [BOW] approach. Burnap and Williams [6] introduced the idea of *othering* language as a useful feature for HS classification. Othering language is the idea of differentiating groups of with "us" versus "them" rhetoric and it has long been observed in discussions surrounding racism and HS. An example of this could be the phrase "send them home". This phrase does not include any known HS words or aggressive language but based on the context one can infer the intent of the author. The authors identified several othering terms and utilized them in their classification task. Their work lends credence to the idea that HS discourse is not limited to the presence or absence of a fixed set of words, but is instead related to the context in which it appears. The idea of out-of-dictionary HS words is a key issue in all related classification tasks and this work provides us with the basis and motivation for constructing a dynamic method for identifying these words.

In most of the works, character *n*-grams were utilised and it represents one of the best and easiest to implement approaches to hate speech detection, often outperforming more sophisticated models. Waseem and Hovey [7] outlined their use along with other Twitter user level data including inferred gender and geolocation. In their follow up work, Waseem [8] speaks to the impact that annotators have on the underlying classification models. This work is particularly critical to the domain of hate speech classification and their experimental results show the difference in model quality when using expert versus amateur annotators. They reported inter-annotator agreement of $K = 0.57$ among amateur annotators. For the expert annotators they reported $K = 0.34$ for the majority vote and $K = 0.70$ for full agreement. The low scores indicated that hate speech annotation is a difficult task and represents a significant and persistent challenge.

Djuric et al. [9] adopted the *paragraph2vec* [a modification of the *word2vec* algorithm for unsupervised learning of continuous representations for larger blocks of texts] approach for classifying user comments as being either abusive or clean. This work was extended by Nobata et al. [10], the most comprehensive work to date. Here the authors made use of features from N-grams, Linguistic, Syntactic and Distributional Semantics. These features form their model, *comment2vec*, where each comment is mapped to a unique vector in a matrix of representative words. The authors used joint probabilities from word vectors to predict the next word in a comment and comment vectors to predict the next word given many contexts sampled from the comment. The other major contribution was the evaluation of their model over time and across domains. In both works the authors used data supplied by Yahoo! Since our work focuses on learning the different contexts in which words appear, we utilize neural embedding approaches with *fasttext* by Bojanowski et al. [11] and *dependency2vec* by Levy and Goldberg [12].

Finally, Magu et al. [13] present their work on detecting hate speech code words. Their work focused on the manual selection of hate speech code words, that is, words that are used by communities to spread hate content without being explicit in an effort to evade detection systems. The authors start with a fixed set of code words and collect and annotate tweets where those words appear for their classification system. These code words have an accepted meaning in the regular English domain, the users in these communities exploit this in order to confuse people who may not understand their hidden meaning. A few examples of these are "*googles*", "*skypes*", and "*butterfly*". It is at this point where we continue the work and propose our method for dynamically identifying new code words. All of the previous studies referenced here utilize either an initial bag of words [BOW] and/or annotated data. The general consensus that the authors come to is that a BOW approach alone is not sufficient. Furthermore, if the BOW remains static then trained models would struggle to classify less explicit HS examples, in short, we need a dynamic BOW. In line with previous studies, we attempt to advance the work by incorporating the different types of textual context that can be derived from as our core features for surfacing new words. This context covers both the topical and functional context of the words being used, the difference will be covered in the methodology. The aim of our work is to dynamically identify new *code words* that are introduced into the corpus and to minimize the reliance on a BOW and annotated data.

III. Hate Speech and Context

Firstly, we must define our assumptions about hate speech and the role that context takes in our approach. Ultimately, we desire to utilise models that can create word representations of *relatedness* and *similarity*. We introduce the various types of context and provide the motivation that lays the ground work for our methodology.

While there exists words or phrases that are known to be associated with hate speech, it can often be expressed without any of these keywords. Additionally, it is difficult for human annotators to identify hate speech if they are not familiar with the meaning of words or any context that may surround the text. These issues make it difficult to identify hate speech with Natural Language Processing [NLP] approaches. We first provide a qualitative definition of hate speech in 3.1 after which we will define the features that will provide a quantitative definition of hate speech for the purposes of our work.

Definition 3.1: (Hate Speech) is text that advocates for any national, racial or religious hatred that constitutes incitement to discrimination, hostility or violence; regardless of the words in use. This includes known hate speech words and phrases in addition to any other combination of words that can be used that meet the requirements outlined.

With this definition in mind, consider the following example, *"Anyone who isn't white or christian does not deserve to live in the US. Those foreign primitives should be deported."* This example fits our definition of hate speech but it would likely be missed by a classifier trained with word collocation features, as it is a subtle and rarer example that does not contain any words strongly associated with hate speech, a problem highlighted by Nobata et al.[10]. We can infer that "primitives" is being used as a code word here and we can also infer possible words that are both similar and substitutable. Using word *similarity* and word *relatedness* features we can infer hate speech usage and this example represents the types of contextual word representations that we wish to identify with our model.

A. Neural Embeddings and Context

As previously mentioned, we desire models that align words into vector space in order to get the neighbours of a word under different uses. These models are referred to as Neural Embeddings; the following definitions and examples provide illustration.

Definition 3.2: (Neural Embedding) refers to various NLP techniques used for mapping words or phrases to dense vector representations. They enable efficient computation of semantic similarities of words based on their distribution in the underlying language corpus. The core idea is based on the theory of Distributional Hypothesis by Harris [14] which states that *"words that appear in the same contexts share semantic meaning"*. In the domain of Neural Embeddings this means that a word will share characteristics with the words that are typically its neighbours in a sentence.

Definition 3.3: (Cosine Similarity) is a standard measure in the domain of Neural Embeddings used to measure how similar two vectors are, it will hereafter appear as sim. Values range from 0 to 1 with 1 indicating equality.

Neural Embedding models represent words in vector space. As mentioned previously, calculating vector (and thus word) similarity is done with cosine similarity. Each word representation will have a given sim score to another word that exists in the model.

Definition 3.4: (similarByWord) refers to the function for ranking words by sim. Given a target word w, an embedding model $\mathcal{E}$, and a specified $topn$ value we find the $topn$ most sim words in $\mathcal{E}$, sorted from highest to lowest cosine similarity. $simByWord$ will be used to reference this function hereafter.

While most Neural Embeddings operate in the same fundamental way, the distinction comes from the input (hereafter referred to as *context*) that they make use of. We introduce *topical context* and *functional context* as key concepts that will influence our features.

Definition 3.5: (Topical Context) is the context used by conventional word embedding approaches that utilize a bag-of-words in an effort to rank words by their domain similarity. A word embedding model might tell us that the words *orange* and *apples* share a high sim because they typically associate with each other. This is best demonstrated in the famous NLP example *v(king) − v(man) + v(woman) ≈ v(queen)*. We simplify this with the term *relatedness*, to indicate that words that often appear together are related.

Definition 3.6: (Functional Context) describes and ranks words by the syntactic relationships that a word participates in. Levy and Goldberg [12] proposed a method of adapting *word2vec* to capture the syntactic dependencies in a sentence with *dependency2vec*. Intuitively, syntactic dependencies refers to the word relationships in a sentence. A syntactic dependency embedding model might tell us that for the word *Florida*, $simByWord$ words might be *New York, Texas, California*; words that reflect that *Florida* is a state in the United States. The proper terminology is that these words are *cohypernyms*, that is, a word who's meaning is included in that of another word. *New York, Texas, California* are *hypernyms* of *State*. We simplify this with the term *similarity*, to indicate that words that share similar functional contexts are similar to each other.

Topical context reflects words that associate with each other (*relatedness*) while *functional context* reflects words that behave like each other (*similarity*). In our work we wish answer the following: *how do we capture the meaning of code words that we do not know the functional context of?*

Definition 3.7: (Dependency Context) is defined as follows: for a target word w in a parsed tweet with modifiers $m_1, ..., m_k$ and a head h, (m_1, lbl_1),...,(m_k, lbl_k), (h, lbl_h^{-1}), where lbl refers to the type of the dependency relation between the head and the modifier (eg. *nsubj, dobj, amod*) and lbl^{-1}

denotes the inverse-relation. Each dependency context is extracted after storing each tweet in CoNLL-U format. This is adapted from the method outlined by [12] [1]. We provide an intuitive example in Table I.

B. Embedding Types

With the different interpretations of *context* defined we now outline the Neural Embeddings used to model our word representations.

Definition 3.8: (*word2vec*) is a technique introduced by Mikolov et al. [15] for creating Neural Embeddings that considers context to be the window for each word in a sentence. It extracts target words and their surrounding words (given a window size) to predict each context from its target word. Formally, each word $w \in W$ is associated with a vector $v_w \in \mathbb{R}^d$ and similarly each context as $c \in C$ is represented as a vector $v_c \in \mathbb{R}^d$, where W is the words vocabulary, C is the contexts vocabulary and d is the length of the vector or the embedding dimensionality. With vectors as learning parameters, *word2vec* seeks to learn vector representations for both words and contexts such that the dot product $v_w \cdot v_c$ associated with "good" word:context pairs is maximized with the end result being a model that captures word *relatedness*.

Definition 3.9: (Dependency Embedding) or *dependency2vec* is a modification of *word2vec* proposed by Levy and Goldberg [12] that considers as context the Dependency Context outlined in Definition 3.7. The goal of the model is to create learned vector representations which reveal words that are functionally similar and behave like each other, i.e., the model captures word *similarity*. *dependency2vec* operates in the same way as *word2vec* with the only difference being the representation of *context*. The advantage of this approach is that the model is then able to capture word relations that are outside of a linear window and can thus reduce the weighting of words that appear often in the same window as a target word but might not actually be related.

In order to give an intuitive understanding and motivation for the use of both *topical* and *functional* context we provide an example. Consider the following real document drawn from our data:

Skypes and googles must be expelled from our homelands

Table I: The importance of word and dataset context

	skypes	
Clean Texts	skyped	whatsapp
	facetime	line
	skype-ing	snapchat
	phone	imessage
Hate Texts	chat	cockroaches
	dropbox	negroes
	kike	facebook
	line	animals
	Relatedness	**Similarity**

[1] http://universaldependencies.org/docs/format.html

With the example we generate Table I where we assume the existence of embedding models trained on relatively clean text and another trained on text filled with hate speech references. For the *relatedness* columns under both datasets we see that the words are related because of the domain. In short, the word *Skype* appears in the domain of internet technologies because it has often occurred with these words in the corpus. We see the same effect for *similarity* under Clean Texts. However, we when looking at *similarity* under the Hate Texts we can infer author is not using Skype in its usual form. The *similarity* columns gives us words that are functionally similar. We do not yet know what the results mean but anecdotally we see that the model returns groups of people and it is this type of result we wish to exploit in order to detect code words within our datasets.

It is for this reason that we desire Neural Embedding models that can learn both word *similarity* and word *relatedness*. We propose that this can be used as an additional measure to identify unknown hate speech *code words* that are used in similar *functional contexts* to words that already have defined hate speech meanings.

IV. Methodology

A. Overview

The entire process consists of three main steps:

- Creating Neural Embedding models that capture word relatedness and word similarity.
- Using graph expansion and PageRank scores to bootstrap our initial HS seed words.
- Enriching our bootstrapped words to learn out-of-dictionary terms that bare some hate speech relation and behave like code words.

The approach will leverage existing research that confirmed the utility of using hate speech blacklists, syntactic features, and various neural embedding approaches as illustrated in Fig. 1. We provide a brief overview of syntactic word dependencies, the different types of context, and finally, how they can be utilised to identify possible code words.

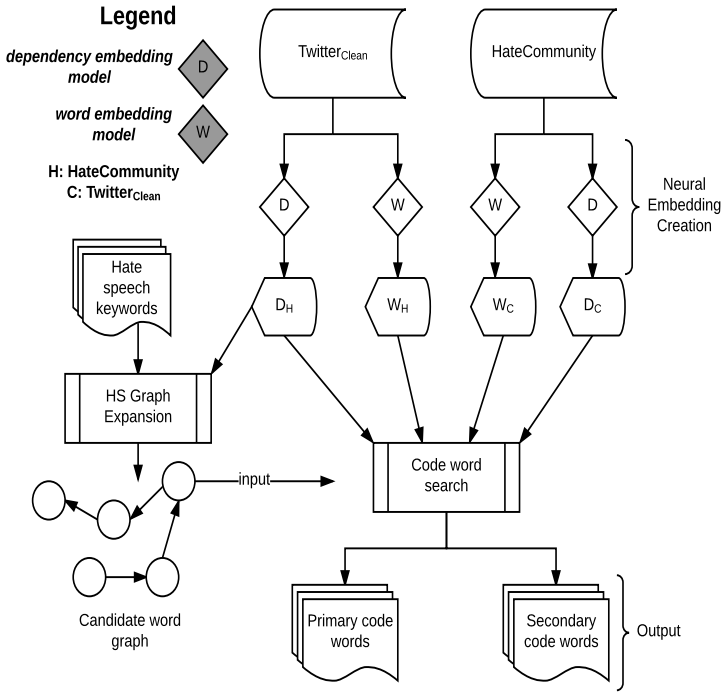


Figure 1: Framework Overview

The process for creating $HateCommunity$ and $Twitter_{clean}$ are outlined in Fig. 2 and Fig. 3 respectively while Fig. 5 details the Neural Embedding Creation. Definition 4.7 highlights the HS Graph Expansion step and finally, Section IV-D provides the rationale behind the Code word search process.

Definition 4.1: (Hate Speech Keywords) is defined as a set of words $\mathcal{H} = \{h_1, h_2, h_3, ..., h_n\}$ as the set of words typically associated with hate speech in the English language. These words were collected from a list maintained by the HateBase organization[2] as used by Nobata et al. [10].

B. Data Collection and Embedding Creation

A key part of our method concerns the data and the way in which it was collected and partitioned, as such it is important to first outline our method and rationale.

Our motivation for creating subsets of the data is two fold:

1) We suspect that there exists words that can take on a vastly different meanings depending on the way in which they are used, that is, they act as code words under different circumstances.
2) Creating a model that align words into vector space allow for the extraction of the neighbours of a word under different uses.

At the outset we had the suspicion that there exists communities of users on Twitter that share a high proportion of hate speech content amongst themselves. Human beings are social animals and so it would follow like most other group, these communities would want to share writing or other content that they produced. We are of the belief that new hate speech code words are created by these communities and that if there was any place to start checking for code words it would be at the source. We began the search by referencing the Extremist Files maintained by the Southern Poverty Law Center[SPLC][3], a US nonprofit legal advocacy organization that focuses on civil rights issues and litigation.

The SPLC keeps track of prominent extremist groups and individuals within the US. It was there that we came across several websites that are known to produce extremist and hate content, most prominent of these being DailyStormer[4]. The articles on this website are of a White Supremacist nature and are filled with references that degrade and threaten non white groups, as such, it serves as an ideal starting point for our hate speech data collection.

We crawl articles from this website, storing the URL as a unique identifier, as well as the author of the article. We then use this list of authors for a manual lookup in order to tie the article author to their Twitter profile. We were not able to identify the profile of each author as we suspect that our list might include pseudo-names. With the Twitter profile IDs we collected the followers of these accounts and their tweets, thus building a network of users.

Definition 4.2: ($HateCommunity$) refers to our dataset which consists of the article content and titles previously mentioned in addition to the historical tweets of users within the network of author followers. The process for generating this dataset can be seen in Fig. 2.

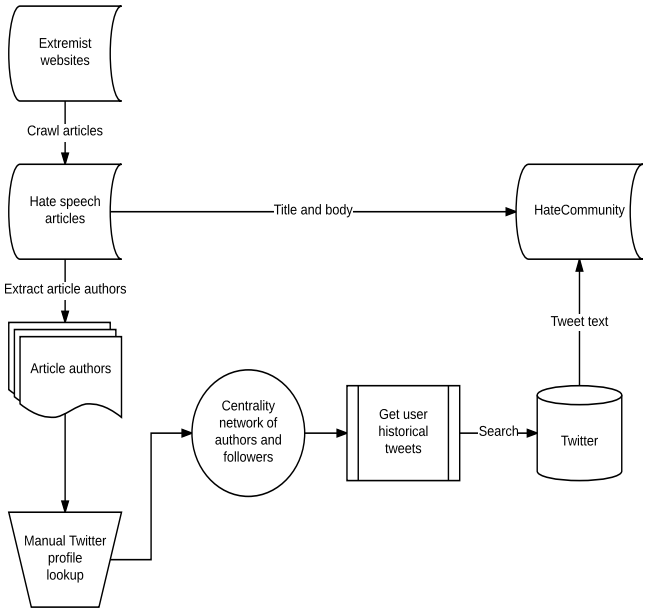


Figure 2: Process outline for $HateCommunity$ creation

[2]We used lists scraped from http://www.hatebase.org/

[3]https://www.splcenter.org/

[4]https://www.dailystormer.com/

Twitter offers a free 1% sample of the total tweets sent on the platform and so we consider tweets collected in this manner to be a best effort representative of the average. To that end we define the following datasets:

Definition 4.3: ($Twitter_{clean}$) refers to our dataset collected without tracking any specific terms or users, only collecting what Twitter returned, free from the bias of collecting data based on keywords. We filter and remove any tweet that contains a word $w \in \mathcal{H}$. The process for generating this dataset can be seen in Fig. 3.

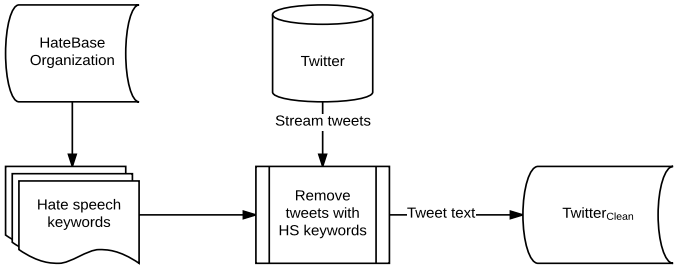


Figure 3: Process outline for $Twitter_{clean}$ creation

Definition 4.4: ($Twitter_{hate}$) refers to our dataset collected using $\mathcal{H}$ as seed words to collect tweets where they appear. The process for generating this dataset can be seen in Fig. 4.

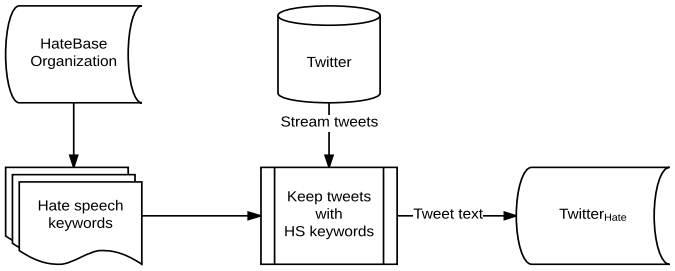


Figure 4: Process outline for $Twitter_{hate}$ creation

One interesting consideration that was initially overlooked was the intersectional nature of the words that surround hate speech. These words can be used for a variety of topics that don't relate to hate speech and as a result of this we found that our $Twitter_{hate}$ was filled with tweets that relate to pornography. We developed a filtering approach that utilized word *n*-grams in order to filter out these tweets, we will make our pipeline and our filtering list publicly available.[5]

[5]https://github.com/IDEA-NTHU-Taiwan/porn_ngram_filter

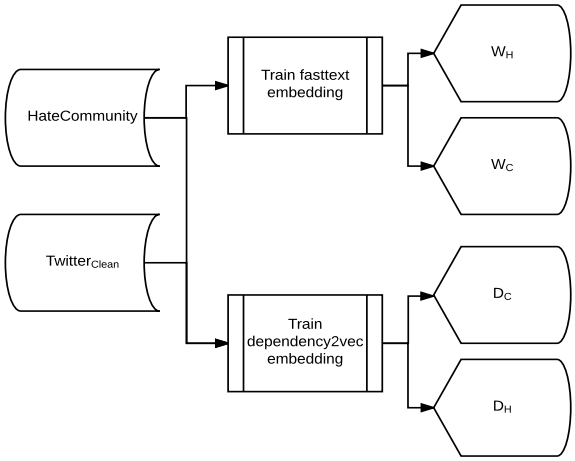


Figure 5: Process outline for creating Neural Embedding models

Our intuition is that we can model the *topical* and *functional* context of words in our hate speech dataset in order to identify out-of-dictionary hate speech code words. For our $HateCommunity$ and $Twitter_{hate}$ datasets we create both a Word Embedding Model and a Dependency Embedding Model as illustrated in Fig. 5. We refer to these as $\mathcal{W}_C$, $\mathcal{W}_H$, $\mathcal{D}_C$, and $\mathcal{D}_H$. We utilized *fasttext* [11] to create our Word Embedding models as it is the current state of the art method for learning word representations. It produces similar output to *word2vec* and has the advantage of capturing words that occur rarely owing to its character level architecture.

Table II: Notations

Notation	Description
$\mathcal{G}$	a contextual graph built with output from $\mathcal{E}$
$\mathcal{D}$	a learned dependency embedding model
$\mathcal{D}_C$	a learned $dep2vec$ model trained on $Twitter_{clean}$
$\mathcal{D}_H$	a learned $dep2vec$ model trained on $Twitter_{hate}$
$\mathcal{E}$	a learned embedding model of type $\mathcal{W}$ or $\mathcal{D}$
$\mathcal{E}_{vc}$	a stored vocabulary for a given embedding model
$HateCommunity$	a dataset subset as outlined in Definition 4.2
$\mathcal{H}$	a set of words associated with hate speech
$\mathcal{G}_{relatedness}$	a word relatedness graph built with output from $\mathcal{W}$
$\mathcal{G}_{similarity}$	a word similarity graph built with output from $\mathcal{D}$
$Twitter_{clean}$	a dataset subset as outlined in Definition 4.3
$Twitter_{hate}$	a dataset subset as outlined in Definition 4.4
$\mathcal{W}$	a learned word embedding model
$\mathcal{W}_C$	a learned word embedding model trained on $Twitter_{clean}$
$\mathcal{W}_H$	a learned word embedding model trained on $Twitter_{hate}$

C. Contextual Graph Expansion

The idea for finding candidate code words is based on an approach that considers the output from the $topn$ word list from our 4 embedding models, given a target word w. Recall that the embedding models represent words as vectors, thus we can use the function outlined in Definition 3.3 to calculate the cosine similarity between vectors and use that as a means of ranking the distance between vectors and thus, the underlying word. The proposed algorithm makes use of directed graphs for an initial seed word expansion and throughout the search process.

As input we accept a list of words and returns out-of-dictionary words along with their contextual representations. Each word is placed into primary and secondary buckets, respectively representing words that may be very tightly linked to known hate speech words and those that have a weaker relation.

Considering that our model attempts to identify out-of-dictionary words that can be linked to hate speech under a given context, we first need to define a way to reduce the number of words that we need to check. To achieve this, we devised a graph construction methodology that builds a weighted directed graph $\mathcal{G}$ of words and their $topn$ output from $simByWord$ (Definition 3.4). In this way, we can construct a graph that models word *similarity* or word *relatedness*, depending on the embedding model we utilize. This graph takes on several different inputs and parameters throughout the algorithm, as such we define the general construction.

Definition 4.5: (Contextual Graph) is a weighted directed graph $\mathcal{G}$ where each vertex $v \in V$ represents a word $w \in seed\_input$. Edges are represented by the set E. The graph represents word *similarity* or word *relatedness*, depending on the embedding model used at construction time.

In all variations of the graph defined in Definition 4.5 we apply a weighting boost to edges from $\forall v \in V$ that share an initial proximity to any $v \in \mathcal{H}$, as described in 4.7. We desire a means of ranking out-of-dictionary words in a graph where some of the vertices are known hate speech keywords. PageRank [16] is suitable for this task as it allows us to model known hate speech words and words close to them as *important links* that pass on their weight to their their successor vertices, thus boosting their importance score. In this way we are able to have the edges reflect that words which are successor of a known hate speech word should get a boost that reflects a higher relevance in the overall graph. Edges are attached as follows, with the weighting scheme outlined in 4.8.

Definition 4.6: (edge) For a pair of vertices (v_1, v_2) an edge $e \in E$ is created if v_2 appears in the output of $simByWord$, with v_1 as the input word. Using the example from Table I with v_1 = *skype*, edges would be created to the vertices corresponding to [*negros, cockroaches, mormons, antifas*]. There are 2 weighting schemes that we employ.

Definition 4.7: (boost) During the construction of any *contextual graph* we do a pre-initialization step where we call $simByWord$ with a given $topn$ for $\forall w \in \mathcal{H}$ if $w \in \mathcal{E}_{vc}$. Recall that $\mathcal{E}_{vc}$ is the stored vocabulary for the embedding model used during graph construction. The frequency of each word in the resulting collection is stored in $boost$. $boost(w)$ thus returns the frequency of the word w in this initialization step, if it exists.

This step gives us an initial list of words that are close to known hate speech keywords. We do this to assign a higher weighting based on the frequency of a word to appear in a collection generated with the $\mathcal{H}$ seed list. Using only the cosine similarity scores as weight would not allow us to model the idea that hate speech words are the important "pages" in the graph which is the key concept behind PageRank. Concisely, this boosting is done to set known hate speech words as the important "pages" that pass on their weight during the PageRank computation.

Definition 4.8: (weight) Let $frq(v)$ denote the frequency of vertex v in $\mathcal{E}_{vc}$ for the given embedding model, and $sim(v_1, v_2)$ the cosine similarity score for the embedding vectors under vertices v_1 and v_2. The weight wt of e(v_1, v_2) is then defined in Equation 1:

$$wt(v_1, v_2) = \begin{cases} \log(frq(v_1)) \times boost(v_1) + sim(v_1, v_2) & \text{if } v_1 \in boost \\ sim(v_1, v_2) & \text{if } v_1 \notin boost \end{cases} \quad (1)$$

With the prerequisite definitions in place we now outline our algorithm for building an individual word contextual graph in Algorithm 1. Intuitively, the algorithm accepts a target word and attaches edges to vertices that appear in the results for $simByWord$. We then collect all vertices in the graph and repeat the process, keeping track of the vertices that we have seen. Note that $depth$ specifies the number of times that we collect current graph vertices and repeat the process of appending successor vertices. A $depth$ of 2 indicates that we will only repeat the process for unseen vertices twice.

Algorithm 1 buildGraph

Input: $w, \mathcal{E}, depth, boost, topn$
Output: $\mathcal{G}$

```
seen_vertices = ∅
G = empty directed graph
predecessor_vertices = simByWord(w, topn)
for vertex:p in predecessor_vertices do
    G += add_edge(w, p, wt(w, p))
end for
seen_vertices += w
for i in range(1, depth) do
    curr_vertices ← G.vertices()
    for vertex:v in curr_vertices do
        if v ∉ seen_vertices then
            successor_vertices = simByWord(v, E, topn)
            for each vertex p in predecessor_vertices do
                G += add_edge(v, p, wt(v, p))
            end for
            seen_vertices += v
        end if
    end for
end for
return G
```

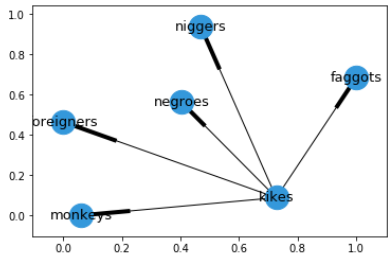
Definition 4.9: (Composed Graph) A composed graph is union of contextual graphs [4.5] created from a lists of words, with a graph being created for each word in the input list. Equation 2 defines the method for generating boost values as defined in 4.7.

$$boost = \bigcup_{h \in \mathcal{H}} simByWord(h, \mathcal{E}, topn) \quad (2)$$

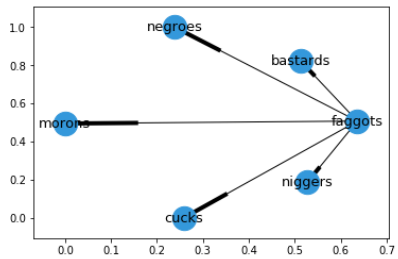
where $topn$ is set to a sufficiently large value. $boost(w)$ then returns the frequency of word w in the set. The formula for creating the composed graph from a list of words then becomes:

$$\mathcal{G} = \bigcup_{w \in word\_list} buildGraph(w, \mathcal{E}, depth, boost, topn) \quad (3)$$

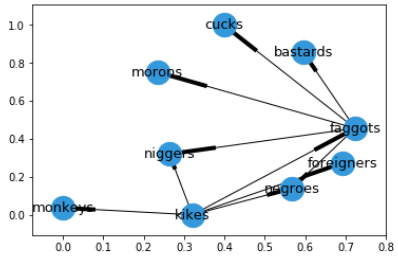
The graphs are constructed and merged on a per word basis. We present a visualization of this process with a composed graph consisting of two words, $topn = 5$, and $depth = 1$.



(a) Graph $\mathcal{G}_1$, built from word$_1$



(b) Graph $\mathcal{G}_2$, built from word$_2$



(c) Composed graph $\mathcal{C} = \mathcal{G}_1 \cup \mathcal{G}_2$

As mentioned at the outset, we need to provide a way of generating candidate words that we can utilize as input for our code word search. We initially explored several methods for doing this, including searching over the entire vocabulary of our partitioned dataset. This did not yield meaningful results and so we settled on the idea of first constructing a contextual graph from $\mathcal{H}$ then using PageRank for ranking and the document frequency df for a given word w. We subsequently remove all known hate speech words from the output.

$$df = \frac{doc\_count(w)}{N} \quad (4)$$

where N is the total number of documents in a given dataset.

Our assumption is that if a word w in our $\mathcal{H}$ Graph is frequently used as a code word, then it should representing have a higher df in $HateCommunity$ over $Twitter_{clean}$. This assumption is supported by plotting the frequencies and observing that most of the words in the $\mathcal{H}$ graph have a high df in $HateCommunity$. We perform several frequency plots and the results confirm our assumption. We therefore use df as our cut off point, ignore words that have a higher df in $Twitter_{clean}$.

Definition 4.10: (HS PageRank Seed) $\mathcal{G}_{hate\_similarity}$ is a contextual graph built using $\mathcal{H}$. We opted to use *similarity* model over *relatedness*. For the PageRank scores we set $d = 0.85$ as it is the standard rate of decay used for the algorithm. $PR = PageRank(\mathcal{G}_{hate\_similarity}, d = 0.85)$. We then trim PR as outlined in Equation 5.

$$\begin{cases} keep(w) & \text{if } df(w \in HateCommunity) > df(w \in Twitter_{clean}) \\ remove(w) & \text{if } df(w \in HateCommunity) < df(w \in Twitter_{clean}) \end{cases} \quad (5)$$

Finally, we further refine our seed list, by building a new graph using the trimmed $PR + \mathcal{H}$, computing a revised PR on the resulting graph. To be clear, only the word in this list and not the actual scores are used as input for our codeword search.

D. Contextual Codeword Search

With our PageRank list as input, we present our method for dynamically generating contextual word representations and the logic for determining if a word acts as a hate speech code word or not. We discuss our concept of primary and secondary buckets, define contextual representation, and finally wrap up with the core of our method.

The issue with code words is that they are by definition secret or at best, not well known. Continuing with the examples of *Skype* and *Google* we previously introduced, if we were to attempt to get related or even similar words from a Neural Embedding model trained on generalized data, it is unlikely that we would observe any other words that share some relation to hate speech.

However, it is not enough to do the same on a Neural Embedding model trained on data that is dense with hate speech. The results might highlight some relation to hate speech but we would know nothing of the frequency of use, in short, we need to have some measure of the use of a word in the general English vocabulary in order to support the claim that these words can also act as hate speech code words. It is for this reason that we propose a model that builds a contextual representation of a word that includes *similarity, relatedness* and frequency of use, all drawn from the differing datasets $HateCommunity$ and $Twitter_{clean}$. Our method not only highlights candidate code words but also determines the strength of the relationship they may have to hate speech with the primary and secondary level categories.

Definition 4.11: (getContextRep) At the core of the method is the mixed contextual representation that we generate for an input word w. The subscript $Hate$ and $Clean$ refer to $HateCommunity$ and $Twitter_{clean}$ respectively. The process is as follows:

$$cRep(w)_{HateSimilar} = simByWord(w, \mathcal{D}_H, topn) \quad (6)$$

$$cRep(w)_{HateRelated} = simByWord(w, \mathcal{W}_H, topn) \quad (7)$$

$$cRep(w)_{CleanSimilar} = simByWord(w, \mathcal{D}_C, topn) \quad (8)$$

$$cRep(w)_{CleanRelated} = simByWord(w, \mathcal{W}_C, topn) \quad (9)$$

Our codeword ranking algorithm is featured in Algorithm 2.

Definition 4.12: (primaryCheck) accepts a word w, its contextual representation, and $topn$ to determine if w should be placed in the primary code word bucket, returning true or false. Here, primary buckets refers to words that have some strong relation to known hate speech words.

Equation 10 and 11 checks whether the number of number of known hate speech words in the contextual representation for a given word is above the specified threshold th.

$$th\_similarity = \left(th \geqslant \frac{size(HW \bigcap cRep_{HateSimilar})}{topn} \right) \quad (10)$$

$$th\_relatedness = \left(th \geqslant \frac{size(HW \bigcap cRep_{HateRelated})}{topn} \right) \quad (11)$$

$$th\_check = th\_similarity \vee th\_relatedness \quad (12)$$

The second condition, Equation 13, whether w has a higher frequency in $HateCommunity$ over $Twitter_{clean}$.

$$freq\_check = (df(w \in HateCommunity) > df(w \in Twitter_{clean})) \quad (13)$$

By Equation 12 and 13 we get the Equation 14 condition for placing w in the primary code word bucket.

$$primary = th\_check \wedge freq\_check \quad (14)$$

Definition 4.13: (secondaryCheck) accepts a word w and its contextual graph $\mathcal{G}$ and searches the vertices for any $v \in \mathcal{H}$, returning the predecessor vertices of v as a set if a match is found as well as. We check that the set is not empty and use the truth value to indicate whether w should be placed in the secondary code word bucket.

$$secondary = predecessor\_vertices(v \in \mathcal{G} \Rightarrow v \in \mathcal{H}) \quad (15)$$

Algorithm 2 codewordRankup

Input: $wordlist, depth, topn, th$
Output: cw_primary, cw_secondary, $\mathcal{G}$
$\mathcal{G}$ = empty directed graph
$cw\_primary = \emptyset$
$cw\_secondary = \emptyset$
for word:w in $\mathcal{D}_H$ **do**
  $\mathcal{G}_w = buildGraph(w, \mathcal{D}_H, depth, boost, topn)$
  $\mathcal{G} = \mathcal{G} \cup \mathcal{G}_w$
  contextRep = $getContextRep(w, \mathcal{D}_H, \mathcal{D}_C, \mathcal{W}_H, \mathcal{W}_C)$
  primary = $primaryCheck(w, G_w, topn)$
  if primary == true **then**
    cw_primary = $cw\_primary \cup contextRep$
  end if
  if primary == false **then**
    secondary = $secondaryCheck(w, \mathcal{G}_w)$
    if secondary $\neq \emptyset$ **then**
      cw_secondary = $cw\_secondary \cup contextRep$
    end if
  end if
end for
return cw_primary, cw_secondary, $\mathcal{G}$

Additionally, we extract the features in Table III for each returned code word.

Table III: Codeword Features

Notation	Description
alt-rel-words	related words not in HS from $HateCommunity$
alt-rel-words-hate	related words not in HS from $Twitter_{clean}$
alt-sim-words	similar words not in HS from $HateCommunity$
alt-sim-words-clean	similar words not in HS from $Twitter_{clean}$
hs-rel-words	related HS words from $HateCommunity$
hs-rel-words-clean	related HS words from $Twitter_{clean}$
hs-sim-words	similar HS words from $HateCommunity$
hs-sim-words-clean	similar HS words from $Twitter_{clean}$
hate-freq	$df(w) \in HateCommunity$
clean-freq	$df(w) \in Twitter_{clean}$
th	hate word match threshold

V. Experiment Results

A. Training Data

In order to partition our data and train our Neural Embeddings we first collected data from Twitter. Both $Twitter_{clean}$ and $Twitter_{hate}$ are composites of data collected over several time frames, including the two week window leading up to the 2016 US Presidential Elections, the 2017 US Presidential Inauguration, and at other points during early 2017, consisting of around 10M tweets each. In order to create $HateCommunity$ we crawled the websites obtained from the SPLC as mentioned in Section IV.B and obtained a list of authors and attempted to link them to their Twitter profiles. This process yielded 18 unique profiles from which we collected their followers and

built a graph of user:followers. We then randomly selected 20,000 vertices and collected their historical tweets, yielding around 400K tweets. $HateCommunity$ thus consists of tweets and the article contents that were collected during the scraping stage.

We normalize user mentions as *user_mention*, preserve *hashtags* and *emoji*, and lowercase text. The tokenizer built for Twitter in the Tweet NLP[6] package was used. It should be noted that the Neural Embeddings required a separate preprocessing stage, for that we used the NLP package Spacy[7] to extract syntactic dependency features.

B. Experimental Setup

As mentioned previously, we utilized *fasttext* and *dependency2vec* to train our Neural Embeddings. For our Dependency Embeddings we used 200 dimension vectors and for fasttext we utilized 300.

To initialize our list of seed words for our approach we built a contextual graph with the following settings.

1) $\mathcal{D}_H$was used to build a contextual graph based on word *similarity*
2) to generate *boost* [Definition 4.9] we set $topn = 20$
3) $depth = 2$
4) We consider singular and plural variations of each $w \in \mathcal{H}$
5) When adding vertices and edges to the graph we set $topn = 3$

This process for expanding our $\mathcal{H}$ seed returned 994 words after trimming with the rationale from Equation 5.

For the *codewordRankup* [Algorithm 2]:

1) $depth = 2$
2) $topn = 5$
3) $th = 0.2$, where k=$topn$

We set $th = 0.2$ after experiments showed that most words did not return more than 1 known hate speech keyword when checking its 5 closest words. This process return 55 primary and 262 secondary bucket words. It should be noted that we filtered for known $\mathcal{H}$ including any singular or plural variations. An initial manual examination of this list gave the impression that while the words were not directly linked to hate speech, the intent could be inferred under certain circumstances. It was not enough the do a manual evaluation as we needed a way to verify if the words we had surfaced could be recognized as being linked to hate speech under the right context. We saw fit to design an experiment to test our results.

C. Annotation Experiment

We have claimed throughout or work that context is important and we designed an experiment to reflect that. Our aim was to determine if a selection of annotators would be able to identify when a given word was being used in a hate speech context without the presence of known hate speech keyword and without known the meaning of the code words. The experiment featured manually selected code words including 1 positive and 1 negative control word. It is important to have some measure of control as many different works including [8] have highlighted the difficulty of annotating hate speech. The positive and negative samples were designed to test if annotators could identify documents that featured explicit hate speech (positive) and documents that were benign (negative).

We built three distinct experiments where:

1) Documents refer to tweets and article titles.
2) 10 code words were manually selected.
3) Participants were asked to rate a document on a scale of very unlikely (no references to hate speech) to very likely (hate speech). [the scale is mapped the range 0 to 4].
4) $HateCommunity$, $Twitter_{clean}$, and $Twitter_{hate}$ were utilized as the sample pool, randomly drawing 5 documents for each code word (10 word X 5 documents for each experiment).
5) We restricted randomly sample documents from featuring known hate speech. words, apart from the positive sample.
6) Control documents were the same across all three experiments.
7) Direct links were only provided for the experiments drawn from $HateCommunity$ and $Twitter_{clean}$. After completing these experiments, participants were given the option to move on to the $Twitter_{hate}$ experiment.

The experiment was designed to draw for our distinct datasets which would reflect the use of the same word across differing situations and contexts. We obtained 52, 53, and 45 responses for $HateCommunity$, $Twitter_{clean}$, and $Twitter_{hate}$ respectively. The full list can be seen in Table IV.

Table IV: Experiment Selection

Code words
niggers [positive control]
snake
googles
cuckservatives
skypes
creatures
moslems
cockroaches
water [negative control]
primitives

Table V: Experiment Sample

another cop killed and set on fire by googles
@user i'm sick of these worthless googles >>#backtoafrica
strange mixed-breed creatures jailed for striking and killing white woman
germany must disinfect her land. one cockroach at a time if necessary

Table V provides a view of a few of the documents annotators were asked to rate. None of the examples features known $\mathcal{H}$ but it is possible to infer the intent of the original author. The experiment also featured control questions designed to

[6]http://www.cs.cmu.edu/ ark/TweetNLP/
[7]https://spacy.io/

test if participants understood the experiment, we provided 5 samples that featured the use of the word *nigger* as positive for hate speech and *water* as negative for hate speech. An overwhelming majority of the were able to correctly rate both control questions, as can be seen in Figs. 8 and 7.

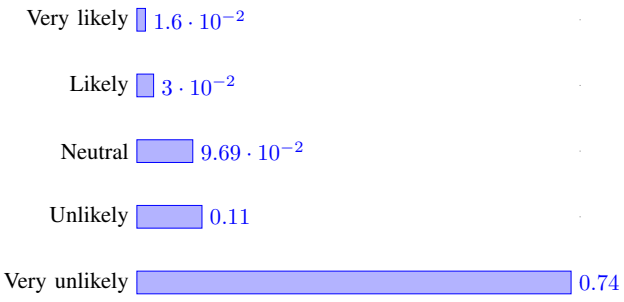


Figure 7: Aggregate percentage for control word "water". Negative for Hate Speech

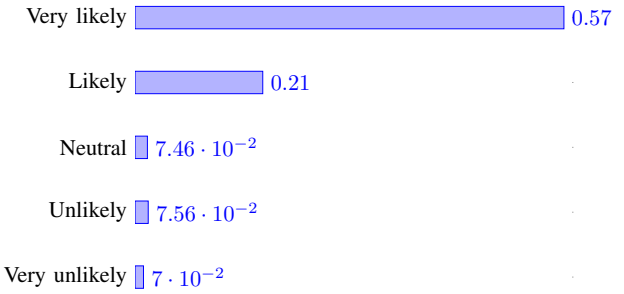
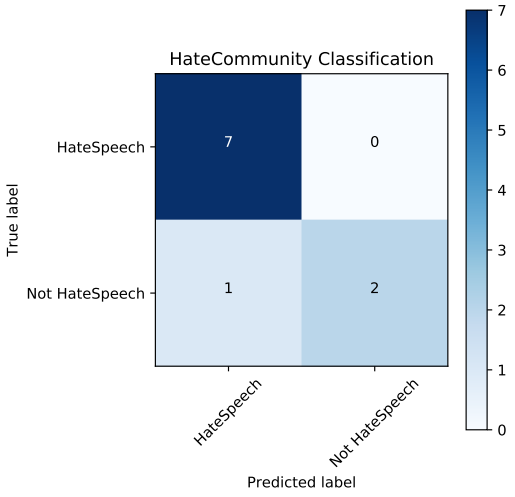


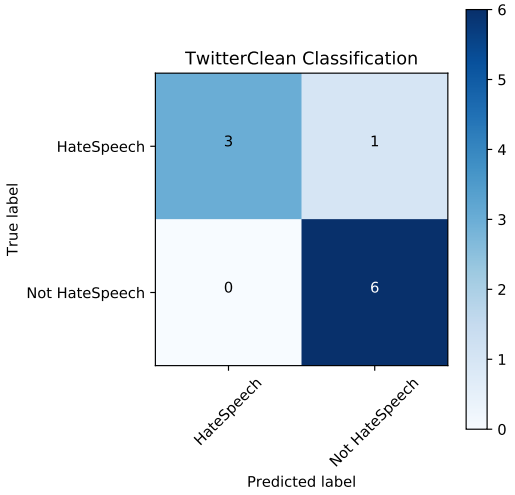
Figure 8: Aggregate percentage for control word "niggers". Positive for Hate Speech

To get an understanding of the quality of the data and to facilitate further experiments, we created a ground truth result and aggregated the annotators based on their majority ratings. We first calculated inter annotator agreement with Krippendorff's Alpha which is a statistical measure of agreement that can account for ordinal data. With the majority rankings, we recorded $K = 0.871$, $K = 0.676$ and $K = 0.807$ for $HateCommunity$, $Twitter_{clean}$ and $Twitter_{hate}$.

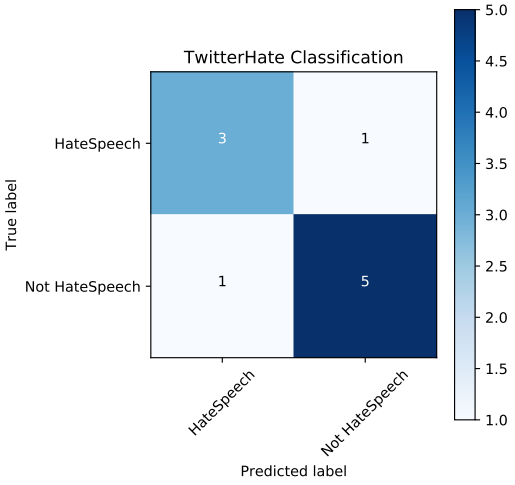
We then moved to calculate precision and recall scores. As we used a likert scale for our ratings, we took ratings that were above the neutral point (2) to as hate speech and ratings below as not hate speech. Interestingly, when taking the majority none of the questions featured the Neutral label as the majority. Our aim for this experiment was to determine if the ratings of the annotator group would reflect hate speech classification when aggregated. The results below show the classification across all 10×5 documents for each experiment (the matrix sums to 10). The Precision and Recall scores can be seen in Figs. 9a, 9b and 9c.



(a) $HateCommmunity$ Confusion Matrix



(b) $Twitter_{Clean}$ Confusion Matrix



(c) $Twitter_{Hate}$ Confusion Matrix

The scores show that when taking the annotators as a single group that they were in line with the ground truth. This gives supports to our claim that it is possible in some cases to infer hate speech intent without the presence or absence of specific words.

Table VI: Aggregate Annotator Classification

		Hate Speech	Not Hate Speech
HateCommunity	Precision	0.88	1.00
	Recall	1.00	0.67
	F1	0.93	0.80
TwitterClean	Precision	1.00	0.86
	Recall	0.75	1.00
	F1	0.86	0.92
TwitterHate	Precision	0.75	0.83
	Recall	0.75	0.83
	F1	0.75	0.83

Further evidence lies in the fact that the results hold up across our three datasets. Table VI shows the F1 scores of 0.93 and 0.86 for $HateCommunity$ and $Twitter_{clean}$ respectively. This result indicates that the annotators were able to correctly classify the usage of the same word under different contexts, from data that is dense in hate speech and data that reflects the general Twitter sample.

One of the ideas that we wanted to verify in the experiment was whether the rankings of the annotators would align with the ground truth. We include the ranking distribution for the $HateCommunity$ experiment results in Table VII and Table VIII. The results compare the majority ranking for each word as well as the percentage against the ground truth.

Table VII: $HateCommunity$ Word:Ranking Distribution

	$HateCommunity$ Results			
	Ground Truth		Annotators	
Words	Label	Agg. Percentage	Label	Agg. Percentage
niggers	Very likely	0.8	Very likely	0.68
snakes	Unlikely	0.4	Neutral	0.26
googles	Very likely	1.0	Very likely	0.41
cuckservatives	Unlikely	1.0	Likely	0.36
skypes	Likely	0.8	Likely	0.3
creatures	Very likely	0.6	Very likely	0.4
moslems	Likely	0.8	Very likely	0.39
cockroaches	Very likely	1.0	Very likely	0.40
water	Very unlikely	1.0	Very unlikely	0.65
primatives	Very likely	0.6	Very likely	0.37

Table VIII: $Twitter_{Clean}$ Word:Ranking Distribution

	$Twitter_{Clean}$ Results			
	Ground Truth		Annotators	
Words	Label	Agg. percentage	Label	Agg. percentage
niggers	Very likely	1	Very likely	0.51
snakes	Very unlikely	0.8	Very unlikely	0.44
googles	Very likely	1.0	Very unlikely	0.84
cuckservatives	Likely	0.6	Likely	0.28
skypes	Likely	0.6	Very unlikely	0.65
creatures	Very unlikely	0.8	Very unlikely	0.77
moslems	Likely	0.6	Very likely	0.35
cockroaches	Very unlikely	0.4	Very unlikely	0.47
water	Unlikely	1.0	Very unlikely	0.78
primatives	Unlikely	0.6	Very unlikely	0.47

VI. Conclusion and Future Work

We propose a dynamic method for learning out-of-dictionary hate speech code words. Our annotation experiment showed that it is possible to identify the use of words in hate speech context without knowing the meaning of the word. The results show that the task of identifying hate speech is not dependent on the presence or absence of specific keywords and supports our claim that it is an issue of context. We show that there is utility in relying on a mixed model of word *similarity* and word *relatedness* as well as the discourse from known hate speech communities. We hope to implement an API that can constantly crawl known extremist websites in order to detect new hate speech code words that can be fed into existing classification methods. Hate speech is a difficult problem and our intent is to collaborate with organisations such as HateBase by providing our expanded dictionary.

References

[1] United Nations General Assembly Resolution 2200A [XX1], "International covenant on civil and political rights," 1966.

[2] Bloomberg, "Disney dropped twitter pursuit partly over image," https://www.bloomberg.com/news/articles/2016-10-17/disney-said-to-have-dropped-twitter-pursuit-partly-over-image/, 2016.

[3] Quartz, "The uk wants new laws to fine google, twitter, and facebook for failing to deal with hate speech," https://qz.com/972583/the-uk-wants-new-laws-impose-serious-fines-on-google-goog-twitter-twtr-and-facebook-fb-for-hate-speech/, May 2017.

[4] W. Warner and J. Hirschberg, "Detecting hate speech on the world wide web," in *Proceedings of the Second Workshop on Language in Social Media*, ser. LSM '12. Stroudsburg, PA, USA: Association for Computational Linguistics, 2012, pp. 19–26.

[5] N. D. Gitari, Z. Zuping, H. Damien, and J. Long, "A lexicon-based approach for hate speech detection," in *International Journal of Multimedia and Ubiquitous Engineering Vol.10, No.4*, ser. IJMUE '15. 20 Virginia Court, Sandy Bay, Tasmania, Australia: Science and Engineering Research Society, 2015, pp. 215–230.

[6] P. Burnap and M. L. Williams, "Us and them: identifying cyber hate on Twitter across multiple protected characteristics," *EPJ Data Science*, vol. 5, no. 1, 2016.

[7] Z. Waseem and D. Hovy, "Hateful Symbols or Hateful People? Predictive Features for Hate Speech Detection on Twitter," *Proceedings of the NAACL Student Research Workshop*, pp. 88–93, 2016.

[8] Z. Waseem, "Are You a Racist or Am I Seeing Things ? Annotator Influence on Hate Speech Detection on Twitter," *Proceedings of 2016 EMNLP Workshop on Natural Language Processing and Computational Social Science*, pp. 138–142, 2016.

[9] N. Djuric, J. Zhou, R. Morris, M. Grbovic, V. Radosavljevic, and N. Bhamidipati, "Hate speech detection with comment embeddings," in *Proceedings of the 24th International Conference on World Wide Web*, ser. WWW '15 Companion. New York, NY, USA: ACM, 2015, pp. 29–30.

[10] C. Nobata, J. Tetreault, A. Thomas, Y. Mehdad, and Y. Chang, "Abusive language detection in online user content," in *Proceedings of the 25th International Conference on World Wide Web*, ser. WWW '16. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2016, pp. 145–153.

[11] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," 2016. [Online]. Available: http://arxiv.org/abs/1607.04606

[12] L. Omer and G. Yoav, "Dependency-based word embeddings," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Short papers)*. ACL, 2014, pp. 302–308.

[13] R. Magu, K. Joshi, and J. Luo, "Detecting the Hate Code on Social Media," in *Proceedings of the Eleventh International AAAI Conference on Web and Social Media (ICWSM 2017)*, 2017, pp. 608–611.

[14] Z. S. Harris, *Distributional Structure*. Dordrecht: Springer Netherlands, 1981, pp. 3–22. ISBN 978-94-009-8467-7

[15] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 3111–3119.

[16] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Technical Report 1999-66, November 1999, previous number = SIDL-WP-1999-0120.